CIND830
Python Programming for Data Science
Lab 8 - Answer Key

1 Arrays

a. The following class defines an array object with a fixed size of elements. However, executing each of the following commands either raises an error or generates an incorrect output. Extend the `iArray` class by defining the proper function(s) to resolve those raised errors.

```
class iArray():
    '''A representation of an array'''
    def __init__(self, capacity, fillValue = None):
        '''Capacity is the static size of the array.
        None object will be placed at each position.'''
        self._items = list()
        for count in range(capacity):
            self._items.append(fillValue)
```

i.
```
a = iArray(5)
len(a)
'''TypeError: object of type 'iArray' has no len()'''
```

```
def __len__(self):
    #Obtaining the size of the array
    return len(self._items)
```

ii.
```
a = iArray(5)
print(str(a).split())
'''Output: ['<__main__.iArray', 'object', 'at',\
   '0x7fd048283310>']'''
```

```
def __str__(self):
    #Representing the array as String
    return str(self._items)
```

iii.
```
a = iArray(5)
a[3] = 2
'''TypeError: 'iArray' object does not support item assignment'''
```

```
def __setitem__(self, index, newItem):
    #Replacing an element in the array
    #Assigning a value to an element
    self._items[index] = newItem
```

iv.
```
a = iArray(5)
print(a[1])
'''TypeError: 'iArray' object is not subscriptable'''
```

```
def __getitem__(self, index):
    #Obtaining an element in the array
    return self._items[index]
```

v.
```
a = iArray(5)
for i in a:
    print(i)
'''TypeError: 'iArray' object is not iterable'''
```

```
def __iter__(self):
    #Traversing elements with loops
    return iter(self._items)
```

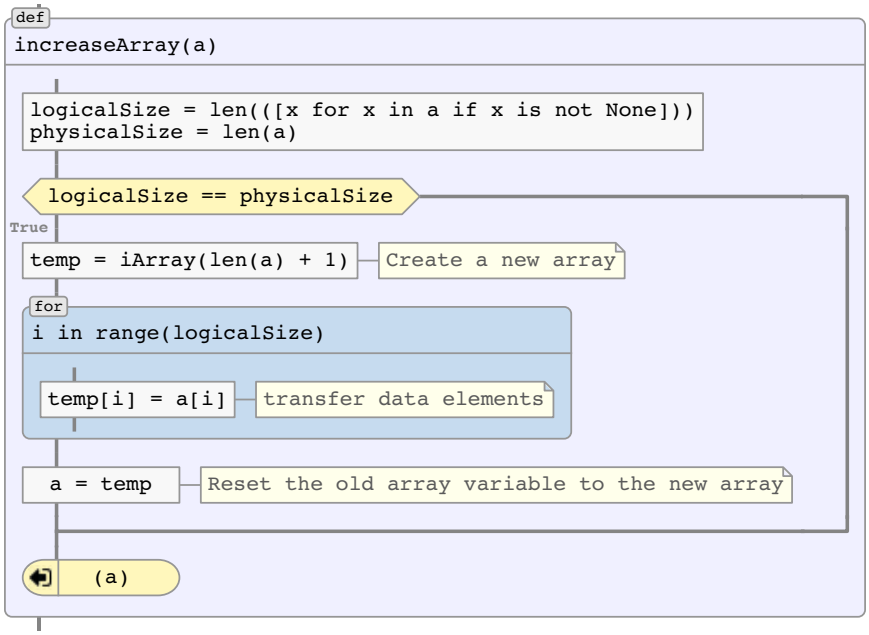
b. The following command gives an `IndexError` indicating that the assignment index is out of range.

```
a = iArray(5)
a[5] = 7
```

To resolve that error, the size of the array needs to be increased. Define a function that resizes a given array using the following steps:

- Check the physical and the logical sizes of the given array.
- Create a new larger array if needed.
- Copy the data from the old array to the new array.
- Reset the old array object to the new array object.

Hint: The following control-flow chart might be of help:

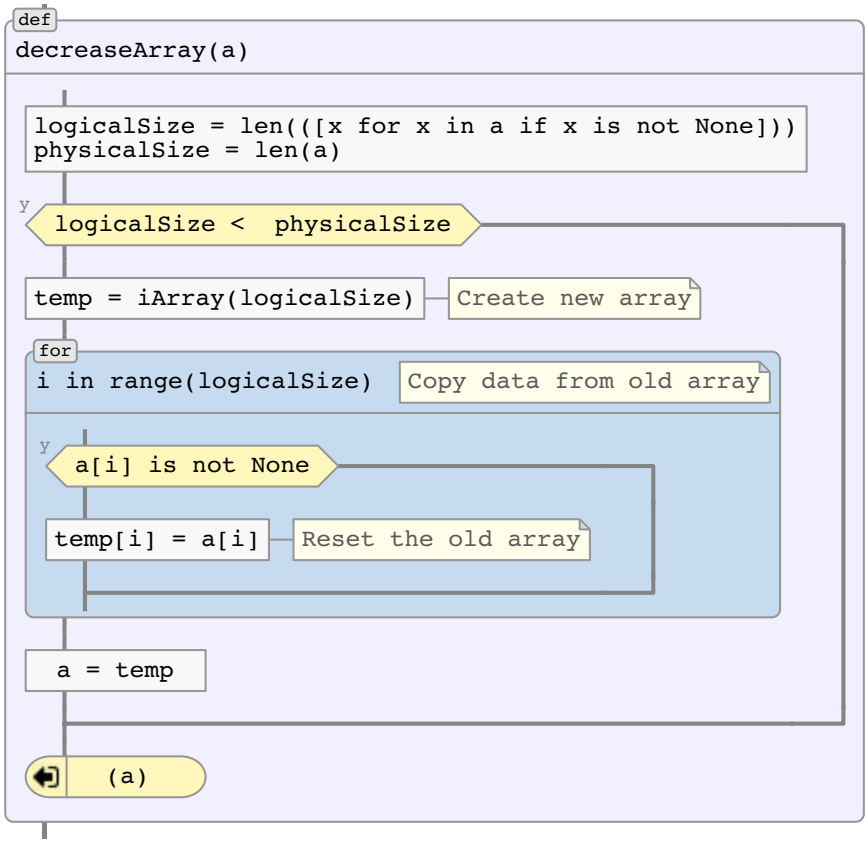



```
def increaseArray(a):
    logicalSize = len(([x for x in a if x is not None]))
    physicalSize = len(a)
    if logicalSize == physicalSize:
        temp = iArray(len(a) + 1)        # Create a new array
        for i in range(logicalSize):
            temp[i] = a[i]               # transfer data elements
        a = temp                         # Reset the old array
    return(a)
```

c. When the logical size of an array is less than the physical size, cells in memory go to waste. To resolve that error, the size of the array needs to be decreased. Define a function that resizes a given array using the following steps:

- Check the physical and the logical sizes of the given array.
- Create a new smaller array if needed.
- Copy the data from the old array to the new array.
- Reset the old array object to the new array object.

Hint: The following control-flow chart might be of help:



```
def decreaseArray(a):
    logicalSize = len(([x for x in a if x is not None]))
    physicalSize = len(a)
    if logicalSize <  physicalSize:
        temp = iArray(logicalSize)       # Create new array
        for i in range(logicalSize):     # Copy data from old array
            if a[i] is not None:
                temp[i] = a[i]           # Reset the old array
        a = temp
    return(a)
```

2 Grids

a. The following class defines a two-dimensional array or a grid object with a fixed size of elements. However, executing each of the following commands either raises an error or generates an incorrect output. Extend the `iGrid` class by defining the proper function(s) to resolve those raised errors.

```
class iGrid():
    '''A representation of a two-dimensional array'''
    def __init__(self, rows, columns, fillValue = None):
        self._data = iArray(rows)
        for row in range(rows):
            self._data[row] = iArray(columns, fillValue)
```

i.
```
g = iGrid(3, 4)
print(g.getRows())
'''AttributeError: \
iGrid instance has no attribute 'getRows'''
```

```
def getRows(self):
    #Obtain the number of rows
    return len(self._data)
```

ii.
```
g = iGrid(3, 4)
print(g.getColumns())
'''AttributeError: \
iGrid instance has no attribute 'getColumns' '''
```

```
def getColumns(self):
    #Obtain the number of columns
    return len(self._data[0])
```

iii.

```
g = iGrid(3, 4)
print(g[0][1])
'''AttributeError: iGrid instance has no attribute '__getitem__
or
TypeError: 'iGrid' object is not subscriptable '''
```

```
def __getitem__(self, index):
    #Indexing with a [row][column] format
    return self._data[index]
```

iv.

```
g = iGrid(3, 4)
print(str(g).split())
'''Output: ['<__main__.iGrid', 'instance', 'at',\
'0x10e3d9870>']'''
```

```
def __str__(self):
    #A string representation of the grid
    result = ' '
    for row in range(self.getRows()):
        for col in range(self.getColumns()):
            result += str(self._data[row][col]) + ' '
        result += '\n'
    return result
```

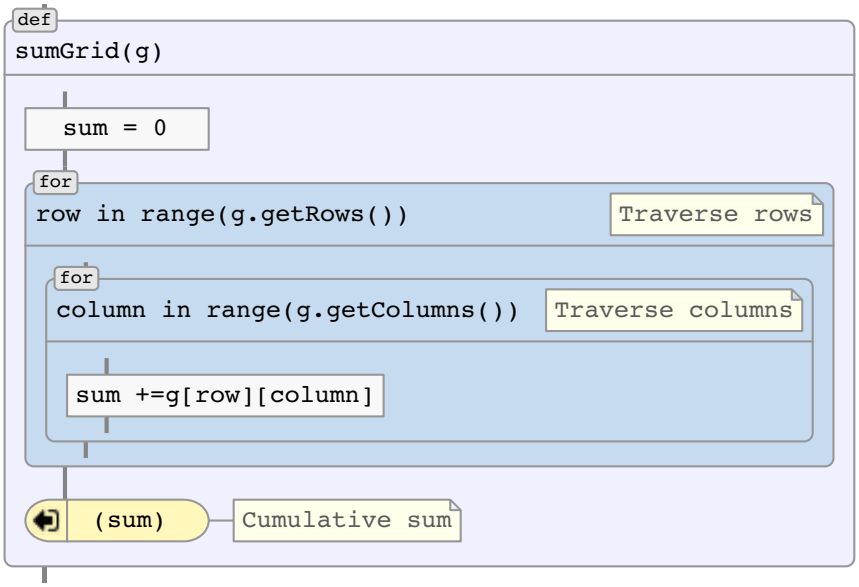
b. Describe the contents of the generated grid after running the following code.

```
iMatrix = iGrid(5,5)
for r in range(iMatrix.getRows()):
    for c in range(iMatrix.getColumns()):
        iMatrix[r][c] = r * c
```

The output is a matrix with five rows and five columns. Each element in the matrix presents the result of the arithmetic multiplication of the element's row index by its column position.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 6 & 8 \\ 0 & 3 & 6 & 9 & 12 \\ 0 & 4 & 8 & 12 & 16 \end{pmatrix}$$

c. Define then test a function that returns the cumulative sum of all the elements inside a given grid. Hint: The following control-flow chart might be of help.




```
def sumGrid(g):
    sum = 0
    for row in range(g.getRows()):              #Traverse rows
        for column in range(g.getColumns()):    #Traverse columns
            sum +=g[row][column]
    return(sum)                                 #Cumulative sum

print(sumGrid(iMatrix))
```

This is the end of lab 8
Tamer Abdou, PhD